Cross-Domain Service Management for Enabling Domain Autonomy in a Federated SOA

Ignacio Silva-Lepe, Isabelle Rouvellou, Rahul Akolkar, Arun Iyengar
IBM T.J. Watson Research Center, 19 Skyline Drive, Hawthorne NY 10532, USA
{isilval,rouvellou,akolkar,aruni}@us.ibm.com

***Abstract*—To tackle SOA projects that span across various boundaries, enterprises are adopting a federated SOA approach in order to manage reuse across service domains. Managing service reuse involves sharing a subset of the services that are provided within a domain and fulfilling references to services required by applications or other services in a domain. While this is a major goal, it is also important for a federated enterprise to enable the autonomy of its multiple service domains. This paper proposes an approach for cross-domain service management that enables domain autonomy and that preserves across domains properties that are taken for granted by services within a domain. We introduce a cross-domain service management capability and show how this capability allows for services to be shared and reused without the need for a federation architect, and how it preserves intra-domain properties, thus enabling domain autonomy in a federation.**

I. Introduction

As the adoption of SOA has taken hold, enterprises have increasingly been moving to SOA projects that span across organizational, functional or geographical boundaries. To tackle these projects, enterprises are, in effect, adopting a federated SOA approach in order to manage reuse across service domains [3], [14], [8]. A service domain, as used in this paper, is a service scope that reflects the underlying organizational, geographical or governance structure of the enterprise. Further, domains in an enterprise use varying SOA implementations that include ESB[1] and registry infrastructure [8].

Managing service reuse involves, on the one hand, sharing a subset of the services that are provided within a domain, for instance a stock quote service, and on the other hand fulfilling references to services required by applications or other services in a domain, for instance a portfolio manager application that requires a stock quote service. To fulfill a required reference, a developer or a domain architect needs to look for the corresponding shared service in other domains in the federation. In order to perform this lookup, the federation's domain membership must be known. Further, if the membership changes, or if the provided services move, the fulfilled reference may become invalid and must be re-fulfilled. In some scenarios, developers or domain architects may be able to rely on a federation architect to fulfill required references; but this only trades a dependency on domain membership knowledge with a requirement on the existence of a federation architect. Managing service reuse also includes exposing, via an indirect federation pattern, a service proxy that handles cross-domain incompatibilities to interact with a service [3]. For instance, a stock quote service may be shared by exposing a proxy that establishes a local control point for customizing access from consumers according to their needs.

[1]An ESB is a distributed infrastructure service that combines message-oriented middleware (MOM), web services, transformation and routing intelligence to connect and coordinate the interaction of diverse services across extended enterprises [2], [5]

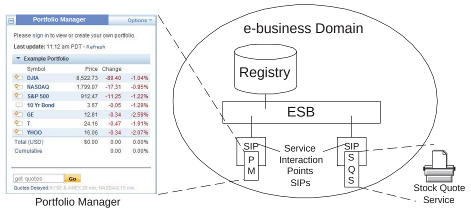
While a major goal is to manage service reuse across domains, it is also important for a federated enterprise to enable the autonomy of its multiple service domains. For the purposes of this paper, we consider as domain autonomy the ability to (1) decide what domain-provided services to share and how, without the need for a central federation authority, and (2) reuse required services without the need for explicit domain membership knowledge or the existence of a federation architect to fulfill required references. This paper proposes an approach for cross-domain service management that enables domain autonomy as defined above and that preserves across domains properties such as location transparency, dynamic selection, and asynchronous connectivity, that are taken for granted by services within a domain.

We believe that, while in some situations a federation architect is desired or appropriate, having a capability that enables domain autonomy makes it possible to obviate the need for a federation architect or for requiring domain membership knowledge in situations where this is not possible or appropriate, without forbidding it in situations where it is.

To this end, we introduce a cross-domain service management capability that (a) allows service requirement and sharing to be expressed independently, (b) disseminates this information via a bilateral protocol of interest and availability messages, and (c) handles proxy management when these messages are matched. We then show how this capability allows for services to be shared and reused without the need for a federation architect, and how it preserves intra-domain properties, thus enabling domain autonomy in a federation. In addition, this capability also allows the removal of a potential bottleneck, represented by a federation architect, that would hinder the scalability of cross-domain service management as the numbers of domains and services within them grow.

II. Intra-domain Service Management

As we alluded to earlier, service federation in an enterprise typically occurs across units of reuse that we refer to as service domains. More precisely, we consider a service domain to



Symbol	Price	Change	
DJIA	8,522.73	-89.40	-1.04%
NASDAQ	1,799.07	-17.31	-0.95%
S&P 500	912.47	-11.25	-1.22%
10 Yr Bond	3.67	-0.05	-1.29%
GE	12.81	-0.34	-2.59%
T	24.16	-0.47	-1.91%
YHOO	16.06	-0.34	-2.07%
Total (USD)	$0.00	0.00	0.00%
Cumulative		0.00	0.00%

Fig. 1. Single Domain Scenario

be a unit of service interaction, visibility and inter-operation. For instance, in SCA [1] a domain defines the boundary of visibility for all SCA mechanisms.

As a motivating example, consider an e-business domain in which a portfolio manager application uses a stock quote service to provide timely stock price information to its users. The e-business domain enables the portfolio manager to find and invoke the stock quote service without paying much attention to its endpoint address or whether or not it is available. Likewise, the stock quote service is able to be deployed and redeployed, and even multiple versions of it made available for differentiated quality of service or performance reasons, without worrying about the impact these changes may have on its consumers. This situation is illustrated in Fig. 1.

Underlying a service domain is infrastructure that includes one or more ESB implementations and a service registry. To provide connection and management to interacting service endpoints, an ESB relies on endpoint interface and destination abstraction. Service providers and consumers interact via a service interaction point (SIP, as in [5], or a service container implementing an abstract endpoint, as in [2]). A SIP provides a destination abstraction layer that decouples the location dependency between provider and consumer. In particular, a SIP represents a service provider to a consumer, but it may delay the determination of the provider's endpoint, by looking it up in a registry, until a request is made. This is one sense in which the ESB incorporates MOM into its architecture.

A service domain uses a service registry to record descriptions of service providers and, to a certain extent, consumers as well. For instance, in [5] "a service registry captures meta-data describing requirements and capabilities of SIPs (for example, interfaces provided or required)". As we shall see, one key form of service requirement description introduced in this paper is the notion of interest as expressed by a service consumer and supported by the service consumer's infrastructure.

Services in a domain can enjoy properties such as dynamic selection, location transparency and asynchronous connectivity. Dynamic selection is a consequence of destination abstraction, where the ultimate target of a service interaction is not determined until the outgoing message is sent. Location transparency refers to the ability of a service provider to change its endpoint address without impacting any of its consumers. Asynchronous connectivity is another consequence of destination abstraction, where the target of a service interaction may not be available at the time the outgoing message is sent. Provided the domain infrastructure has a way to hold on to the message long enough, the sender does not have to be constrained by the availability of the provider.

As we shall see, these properties will be used as a minimal litmus test for our service management automation techniques to enable independent service domains to maintain their autonomy.

III. Cross-domain Service Management

Let us now imagine a situation where the portfolio manager and the stock quote service are located in independent and autonomous domains. For instance, the stock quote service may be provided by a financial services department on a separate domain. This can also occur because a better implementation may be available on a platform provided by a different vendor. This situation is illustrated in Fig. 2. In this case, in order to handle the differences across domains, it is necessary to add a cross-domain proxy to the stock quote service into the domain of the portfolio manager. Specifically, a cross-domain proxy (or proxy, for short) may be encapsulated via a SIP that effectively implements a cross-domain federation pattern[2]. From the point of view of the portfolio manager, the stock quote service proxy (labelled `SQSP` in the figure) can be found and invoked like the actual service.

Adding the stock quote service proxy is a manual operation that also requires adding an entry for the proxy into the e-business domain registry. Now imagine that the IT Services department decides to consolidate service offerings across the enterprise and as a result the stock quote service is relocated to the corresponding domain. In this situation, the stock quote service proxy will need to be updated since it most probably will not know how to communicate with the IT Services domain. This update is again a manual operation. But if we think about it, initially connecting the portfolio manager to the stock quote service, or moving the stock quote service from one domain to the other, should not have any more impact than performing these operations within a single domain. What is needed is a way to automate the steps required to handle the effects of performing these operations.

A federation can be thought of as the loose coupling of service domains, where service connectivity across domains is encapsulated in cross-domain proxies. As such, a federation cannot be assumed to define a single, centralized registry. Also, it is not desirable for each service domain registry to contain entries for every service in every domain in the federation. On the other hand, when services are introduced into a domain or move from one domain to another, connectivity with service consumers must be established and maintained by the insertion or removal of the appropriate cross-domain service proxies. If this connectivity maintenance required manual intervention, then properties such as dynamic selection

[2]Conceptually, a cross-domain proxy is represented by a SIP that is capable of handling the heterogeneity across the ESBs that realize the service domains in question. This is similar to the idea of a bridge that is described in [2] as a way to attach legacy protocols into an ESB, or to interoperate across MOM implementations.

and location transparency would effectively be unavailable at the federation level. In other words, if the introduction and mobility of services also required the manual maintenance of proxies, selection would not be as dynamic and location would not be as transparent for services across domains as they are within the same domain. We introduce cross-domain service management automation that addresses this problem by relying on three elements:

(1) The ability for a service consumer to express and record its requirement for, or interest in, a service satisfying a particular description; this is in addition to the ability of a service provider to advertise and record its availability as satisfying a particular description,

(2) The bilateral dissemination of interest and availability information that are matched at either the consumer end as incoming availability with interest, or at the provider end as incoming interest with availability, and

(3) The cooperative handling of proxy maintenance between a provider and a consumer that automates the connectivity across domains.

Accordingly, each of these elements is realized by three aspects of a service domain: (1) service description maintenance, (2) interest and availability dissemination and matching, and (3) proxy handling automation.

Service description maintenance. Service Management automation relies on service description abstraction to decouple service consumers and providers in the specification of their interest and availability. To that end, the description of a service may be given by a functional interface, non-functional criteria such as quality of service, or (in)formal semantics.

As is customary, when a service is deployed as a provider within the scope of a service domain, the service domain's registry records a description of the service, as well as connectivity information such as endpoint and protocol binding. In our example, let us imagine that the stock quote service is described using a WSDL description that includes interface, binding and service sections. Also, imagine the service assumes its consumers communicate with it using SOAP/JMS. But, since SOAP/HTTP is so common, a converter proxy is deployed with the service, and its WSDL description includes both SOAP/HTTP and SOAP/JMS in its binding section.

Furthermore, a service may also specify information about other services that it requires. For instance, the portfolio manager in our example can specify the requirement for a stock quote service. This can happen at the time the portfolio manager is being authored, which results in a description of the required stock quote service being recorded by the portfolio manager domain's registry. Imagine that the portfolio manager expects to communicate with the stock quote service using SOAP/HTTP. The description of the required stock quote service contains a WSDL description that includes SOAP/HTTP in its binding section to record this as part of the requirement.

The registry maintains such a description of a required service as a service reference. A required service reference can be either fulfilled or unfulfilled. A fulfilled reference

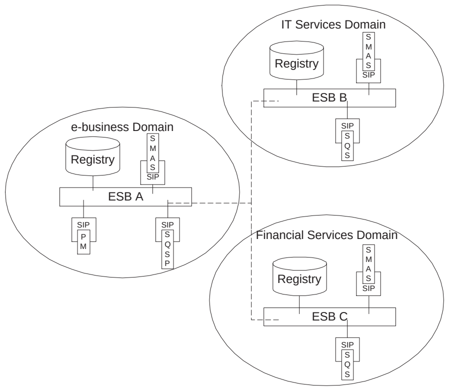


Fig. 2. Multiple Autonomous Domains and Service Management Automation Service

includes, in addition to the description of the required service, the proxy that encapsulates the necessary cross-domain connectivity logic. An unfulfilled reference allows connectivity to be established asynchronously, if and when availability of the required service is received by the service domain. Also, a required service reference may be fulfilled multiple times. This allows for cross-domain dynamic selection. For example, suppose that there are stock quote service providers at both the IT services domain and the financial services domain. The stock quote required service reference used by the portfolio manager may be fulfilled by the availability of both providers, which results in proxies to both services being recorded. This situation is illustrated in Fig. 2, where the `SQSP` SIP is aware of both proxies and is able to select between them.

Interest and availability dissemination and matching. Given the ability of service consumers and providers to express interest and advertise availability, a service domain takes these notifications and exchanges them with other domains; when an exchange results in a match, connectivity can be established by the addition of a suitable proxy. This procedure is embodied in a Service Management Automation Service (SMAS) that is part of a service domain's infrastructure. Fig. 2 shows our example domains also including their respective instances of the SMAS. The SMAS behaves as a representative of either a consumer expressing interest, keeping track of required service references, or of a provider advertising availability, keeping track of provided service entries. The behavior of the SMAS is illustrated in Fig. 3, and it is described as a pair of state machines, one for each of interest and availability aspects of the SMAS. State information is managed via the registry of a given domain. A provided service can be in one of two states, and a required service reference can be in one of three states. State transitions are triggered by either local events, such as a new service being made available, or by the arrival of an interest or an availability message. A registry's state is persistent, so a registry failure can be recovered from its persistent store.

Consider the use case of matching the portfolio manager's interest with the availability of a stock quote service. The port-

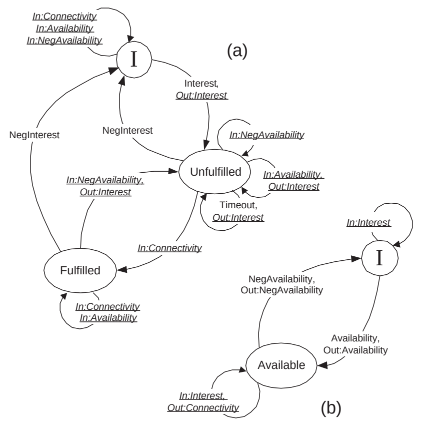


Fig. 3. Service Domain SMAS Behavior

folio manager domain's SMAS takes the interest notification and sends out an interest message to available domains. This is illustrated in Fig. 3(a) with a transition from the initial state to the unfulfilled state. Independently, the financial services department domain's SMAS takes the stock quote service availability notification and sends out an availability message to interested domains. This is illustrated in Fig. 3(b) with a transition from the initial state to the available state. If the portfolio manager domain's SMAS receives an availability message while in the unfulfilled state, it first tries to determine whether the service description in the message matches the stock quote service. If it does then it sends out another interest message, reiterating its intention. When the financial services domain's SMAS receives an interest message, it first tries to determine whether the service description in the message matches the description of an available service. If it does then connectivity can be established.

The matchmaking between service descriptions, which occurs whenever a message (be it interest or availability) is received at a given service domain, involves matching functional, non-functional and informal semantic descriptions. When an interest or availability message arrives at the SMAS, which conveys the description D of the subject service, an entry in the domain's registry is looked up that matches D. For the purposes of this paper, we consider a functional matching of service descriptions. Specifically, we assume the functional portion of a service description to be given by a WSDL description. In a provided service description, as well as in an availability message, this WSDL description is assumed to be complete, in the sense that it includes interface, binding and service sections. For a required service reference, as well as in an interest message, this description may be incomplete, to allow a service consumer to specify only the details that are considered necessary. For instance, the portfolio manager provides a WSDL description that includes an interface section and a binding section that specifies SOAP/HTTP, to indicate its preference of communication protocol. Matching then amounts to determining whether the WSDL description in an arriving interest message (or in a required service reference) is a subset of a provided service description (or of an arriving availability message).

Proxy handling automation. The end goal of cross-domain service management automation is to eliminate the need for manual intervention in the addition and removal of proxies. This is accomplished as a cooperative effort between a provider domain and a consumer domain when interest and availability find each other, or when a service becomes unavailable at a provider domain. In our example, when the financial services domain's SMAS determines that connectivity can be established it sends out a connectivity message. When this connectivity message arrives at the portfolio manager domain's SMAS, and assuming that the service description in the message matches the stock quote service, a transition to the fulfilled state occurs.

A connectivity message conveys a cross-domain service proxy, which includes the logic and mechanism necessary for a service in a foreign domain to communicate with a service in the domain that issues the proxy. When the financial services domain's SMAS issues a connectivity message, it includes the SOAP/HTTP to SOAP/JMS converter proxy to the stock quote service in the message that it sends out. When the portfolio manager domain's SMAS receives the connectivity message, it uses the converter proxy to create the fulfilled service reference that it adds to the registry for the portfolio manager to use. Thus, as we can see, the steps taken to include the converter proxy into the connectivity message, and to add it to the consumer service's domain and registry, effectively automate the manual process of proxy addition.

If and when the stock quote service becomes unavailable at the financial services domain, the SMAS sends out a negative availability message. Receipt of this message at the portfolio manager's domain causes it to transition back to the unfulfilled state, which automates the process of removing the fulfilling proxy.

IV. Enabling Domain Autonomy

In this section we show how the service management automation techniques introduced in section III enable domain autonomy.

First, let us consider service sharing and reuse. When the description of a provided service is added to a domain's registry, and a subsequent availability message is disseminated and matched to a required service reference somewhere in the federation, the provider of the service was able to share it without the need for a central federation authority. Similarly, when the definition of a required service reference is added to a domain's registry, and a subsequent interest message is disseminated and matched to a provided service somewhere in the federation, the interested user of the referred service was able to reuse it without the need for explicit domain membership knowledge or the existence of a federation architect. This way, domain autonomy is enabled with respect to service sharing and reuse.

Note that when service descriptions are added to a domain registry as part of a provided service or a required service reference, it is important to add a description that has a good chance of matching. To this end, we assume that there is a dictionary of standard service templates. This dictionary would be agreed upon by the federation and would evolve over time. A service template includes the service name (e.g. ”Stock”, ”Map”) and a parameter list. A description of what the service should do would also be stored in the dictionary. Users would write service descriptions corresponding to the specifications in the dictionary. Note that implementations may not exist for all of the services in the dictionary. Furthermore, multiple implementations of a given service may exist. When a consumer wants to reuse a service, it uses the dictionary to determine the service template to refer to in his service description. For the purposes of this paper, we focus on template-based service description authoring and leave semantic matching for future work. It may also be the case that services exist that are not in the dictionary. Interest and availability dissemination and matching is also applicable to such a situation. Here, a consumer has prior knowledge of a service whose template is not in the dictionary, which exists in the federation, but whose location is not known by the consumer.

Now, let us revisit intra-domain properties and how they are enabled across domains by service management automation.

Dynamic selection. In a single domain, a consumer SIP determines the target endpoint to invoke using the registry. The registry may contain more than one endpoint for a given service, to allow for load balancing or differentiated quality of service. If this is the case, the consumer SIP can use a load balancing algorithm or application-specific criteria to determine which endpoint to actually use.

In a multiple domain scenario, a consumer SIP looks up a required service reference in the registry. If this reference is fulfilled, it contains a current proxy to connect with the provider service. If the required service reference can be fulfilled multiple times, then the proxy SIP is aware of multiple proxies and can use load balancing or application-specific criteria to determine the proxy to use.

For instance, Fig. 2 shows the stock quote service proxy SIP with a choice to dynamically select connectivity to the stock quote service in the IT services domain or the financial services domain.

Notice that the use of a required service reference, which may be fulfilled multiple times, in conjunction with interest and availability dissemination and matching, and proxy handling automation, allow a consumer SIP to regard dynamic selection across domains as equivalent to dynamic selection within a single domain.

Location transparency. In a single domain, when a provider service changes location, its entry in the registry gets updated with the latest endpoint. Thus, when a consumer SIP determines the target endpoint to invoke, it will always be able to invoke the provided service at its current location.

In a multiple domain scenario, when a provided service changes location from one domain to another, it does so by becoming unavailable in the old domain and becoming available in the new domain. This translates into the proxy to the old domain being removed from the required service reference in the consumer domain, and the proxy to the new domain being added. This in turn means that when the consumer SIP looks up the required service reference, it will always find a proxy to the current domain, modulo any time delays which, as it turns out, may be handled via asynchronous connectivity.

For instance, in Fig. 2, if instead of having a stock quote service at both the IT services and the financial services domains, the stock quote service moved from one domain to the other, then the stock quote service proxy would have a single proxy to the domain where the service resides.

Notice that, via the use of interest and availability dissemination and matching, and proxy handling automation, a proxy in a consumer domain is removed and re-added to its required service reference as a result of the corresponding provider service being removed from an old domain and added to a new domain. Thus, from the point of view of a consumer service and its SIP, the location of a provider service in any given domain becomes transparent.

Asynchronous connectivity. In a single domain, if a provided service is not ready to handle requests, the consumer SIP may be able to block until the provided service becomes ready, or it may provide an asynchronous interface to wait for the provider to become ready without blocking the consumer. If a response is required, the consumer may obtain it from the SIP at a later time.

In a multiple domain scenario, a required service may not be available in any domain at the time of a consumer request. Here, the consumer SIP may be able to block or use an asynchronous interface as well. Furthermore, in this case, the SIP waits for the required service reference to be fulfilled by a connectivity message that results from interest finding availability.

For instance, in Fig. 2, the portfolio manager SIP provides an asynchronous interface that returns immediately upon a request and returns a response if and when connectivity with the stock quote service has been established.

Notice that, via the use of interest dissemination and the subsequent proxy handling automation, a consumer SIP is able to regard unavailability of a provided service in other domains as the lack of readiness of the required service.

This way, by allowing services to be shared and reused without the need for a federation architect, and by preserving intra-domain properties, we can say that the cross-domain service management capability we have introduced enables domain autonomy in a federation. As an added benefit, when service management is automated and a federation architect can be eliminated, connecting and reconnecting any and all clients of any service that is added, removed or relocated in a sizable federation becomes more scalable as the number of domains and services increases.

V. Implementation

A prototype of the service management automation techniques described in this paper is being implemented and integrated with the SOAlive hosted environment [12], which also provides a minimal ESB function.

VI. Related Work

In [10], Rocco et al describe a domain-specific (e.g., DNA sequence) and crawler-based system that discovers candidate services that are relevant to the domain of interest, and that performs service matching via intelligent filtering (with respect to domain-specific description). Our service management automation techniques are not domain-specific. Also, one key aspect of our work is the bilateral nature of the interest/availability protocol[3], where both service providers and consumers broadcast (or multicast) their availability and interest respectively, and when one such message is received and a match occurs, connection is triggered.

The Jini system [6] enables the construction of distributed systems of federated services. In Jini, a lookup service (LUS) is used as a broker. The LUS does not have to be centralized but it does represent an intermediary and it maintains a registry of services. In our case, we don't rely on a registry maintained by an intermediary. Distributed registries communicate to exchange entries of interest. A single broadcast or multicast of interest will contact multiple registries. Only those contacted registries that match the description will send a connectivity message back. This can also result in chained connectivity, as in the case of mediations.

In the Simple Service Discovery Protocol (SSDP) [4], devices advertise services using datagrams sent using IP multicast. In addition, clients interested in accessing services can either wait for announcements or can multicast a search request that forces all devices on the network to send service advertisements. However, in addition to maintaining fulfilled references to required services, a service domain registry also maintains unfulfilled references; this allows a service consumer to have its interest fulfilled asynchronously at a later time, when the provider service becomes available, which is not contemplated by SSDP, among other things.

A review of additional matching techniques can be found in [11]. Policy matchmaking techniques have been proposed using syntactic approaches where policies are compared for structural and syntactic compatibility. For example, in [13], the authors propose a declarative language (extending WS-Policy) for specifying QoS features preferences and conflicts as well as a resolution mechanism that can reason using those specifications and allows components to interoperate. Other approaches have been enhancing description with semantics by creating assertions based on terms from ontologies [7], [9].

VII. Conclusions

We have introduced cross-domain service management automation as a mechanism to enable domain autonomy in a federation. We have seen how services can be shared and reused without the need for a central federation authority or explicit domain membership knowledge. We have also seen how intra-domain properties such as dynamic selection, location transparency and asynchronous connectivity are available across domains. Together, these features of cross-domain service management automation enable domain autonomy in a federation as we have defined it.

This paper focuses on establishing and maintaining cross-domain connectivity via the automated manipulation of proxies. Other equally important cross-domain concerns include service visibility management, and service security and governance. Specifically, interest and availability messages must consider service visibility, connectivity messages must abide by governance policies, and cross-domain proxies must enforce security and access control. Future work in enabling domain autonomy in a federated SOA must of course incorporate these concerns, and we are in the process of evolving our service management automation techniques accordingly.

[3]Rather than using crawling to perform matching.

References

[1] Service Component Architecture Specifications. `http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications`.

[2] David A. Chappell. *Enterprise Service Bus*. O'Reilly, 2004.

[3] Greg Flurry and Marc-Thomas Schmidt. Exploring the Enterprise Service Bus: Part 4: Federated connectivity in the enterprise. `http://www.ibm.com/developerworks/websphere/library/techarticles/0901_flurry/0901_flurry.html`, January 2009.

[4] Y. Goland, T. Cai, P. Leach, Y. Gu, and S. Albright. Simple Service Discovery Protocol/1.0. IETF Internet Draft draft-cai-ssdp-v1-03.txt, October 1999.

[5] Beth Hutchison, Marc-Thomas Schmidt, Dan Wolfson, and Marcia Stockton. SOA programming model for implementing Web services. `http://www.ibm.com/developerworks/webservices/library/ws-soa-progmodel4/`, July 2005. Part 4: An introduction to the IBM Enterprise Service Bus.

[6] Jini. `http://www.jini.org`.

[7] E.M. Maximilien and M.P Singh. A framework and ontology for dynamic web services selection. *Internet Computing, IEEE*, 8:84–93, 2004.

[8] Joseph Natoli. Usage of a SOA 'soft appliance' for Federated SOA. `http://software.intel.com/en-us/blogs/2008/08/15/usage-of-a-soa-soft-appliance-for-federated-soa/`, August 2008.

[9] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. Semantic matching of web services capabilities. In *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web*, pages 333–347, London, UK, 2002. Springer-Verlag.

[10] Daniel Rocco, James Caverlee, Ling Liu, and Terence Critchlow. Domain-specific web service discovery with service class descriptions. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services*, pages 481–488, Washington, DC, USA, 2005. IEEE Computer Society.

[11] Pavel Shvaiko and Jerome Euzenat. A survey of schema-based matching approaches. *Journal on Data Semantics*, 4, 2005.

[12] I. Silva-Lepe, R. Subramanian, I. Rouvellou, T. Mikalsen, J. Diament, and A. Iyengar. SOAlive Service Catalog: A Simplified Approach to Describing, Discovering and Composing Situational Enterprise Services. In *Proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC 2008)*, Sydney, Australia, December 2008.

[13] Eric Wohlstadter, Stefan Tai, Thomas Mikalsen, Isabelle Rouvellou, and Premkumar Devanbu. Glueqos: Middleware to sweeten quality-of-service policy interactions. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 189–199, Washington, DC, USA, 2004. IEEE Computer Society.

[14] Mamoon Yunus. Federated SOA: A Pre-Requisite for Enterprise Cloud Computing. `http://cloudcomputing.sys-con.com/node/1233457`, January 2010.